HERZUM SOFTWARE

# Architectural Reference Models in Middleware Solutions

***SOA, Web Services and Molecular Messengers***
***Keeping Your Architectural Perspective***

Bill Nadal, CTO – Herzum Software
OMG MDA SOA Web Services Conference
March 23rd, 2005, Orlando, FL

HERZUM SOFTWARE

# Overview

- Herzum Software Overview
- Overview of Architectural Styles
- COSM™ Levels of Information Exchange
- COSM™ Interoperability Reference Model
- Evolution of Architectural Styles
- Service Based Approaches
- Service Based Platform (Component Execution Environment)
- Future Middleware and SOA Approaches

# About Herzum Software

- International consulting group, present in US and Europe (Italy, UK, France, Turkey. Partners in Poland, Sweden, South Africa).
- Premier supplier of services for Enterprise Architecture and Technology Strategy, SOA integration and implementation, and agile software manufacturing
- Specialized in tactical and strategic architectural and organizational migration to new technologies and agile outsourcing
- Unique experience in component-based development and Service Oriented Architectures. Ideal partner for jump-starting and following through strategic developments
- Extensive network of strategic alliances with service and product companies
- Capable of handling projects and organizations from startups to Fortune 100

# Herzum Software Offering

A comprehensive set of services including:

- IT Strategy and Enterprise Architecture (small & large enterprises)
- SOA and Enterprise Integration
- Software Architecture and advanced technologies
- Agile Software process definition and rollout
- Extensive curriculum of advanced technology and architectural courses ("*Herzum Software: where architects learn to architect*")
- Mentoring on advanced component-based and Web services technologies
- Software Development
- Agile Outsourcing

HERZUM SOFTWARE

# Architectural Styles

- **Software architecture comes in different "styles". At a high level, examples of main styles are: object-oriented architecture, distributed-object architecture, component-based architecture, service-based architecture, service-oriented architecture**

- **Each style comes with specific architecture characteristics, addresses specific problems, and has limitations addressing other problems**

- **Typically, a good architecture uses a combination of styles**

- **Technologies at times imply or suggest, for example through specific examples, a certain way of applying the technology: the implicit *style* of that technology**
  - **Technologies can be applied very differently from the implicit *style***

# Usage of the *Service* Term

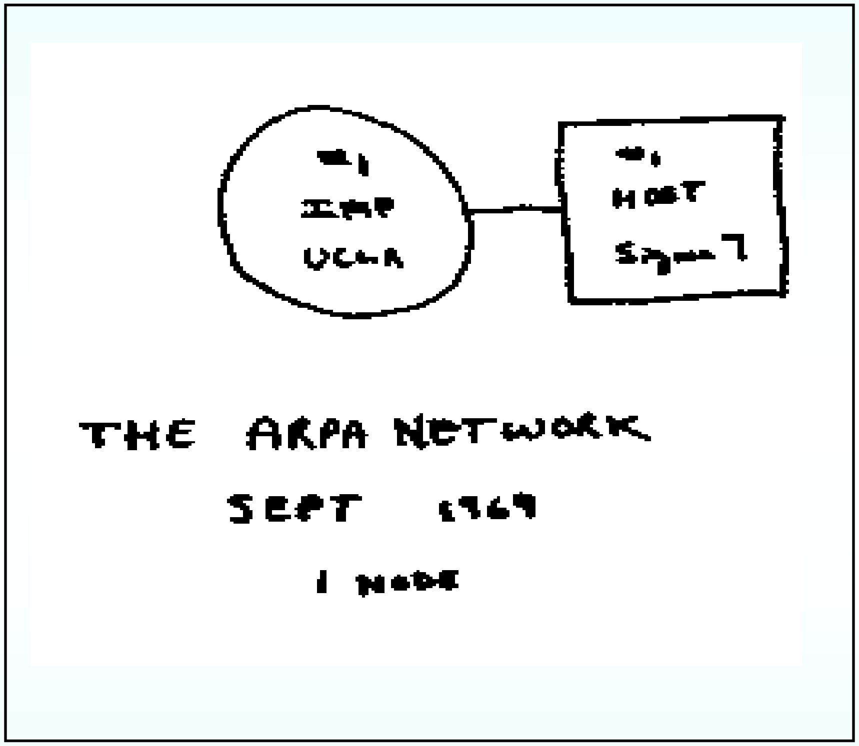
- Technologists and non-technologists both use the term "service"
  - Technologists: an interface that can be called at run-time
  - Non-technologists: an offering provided by a business, e.g. the ability to reserve a flight ( a "business service")
- *Service* will be used in the Technologist sense in this presentation unless otherwise specified
  - Technically speaking, all distributed technologies with an API-like paradigm can be called "service-based"
    - *E.g. RPC, DCE, CORBA, RMI, DCOM are "service-based"*

HERZUM SOFTWARE

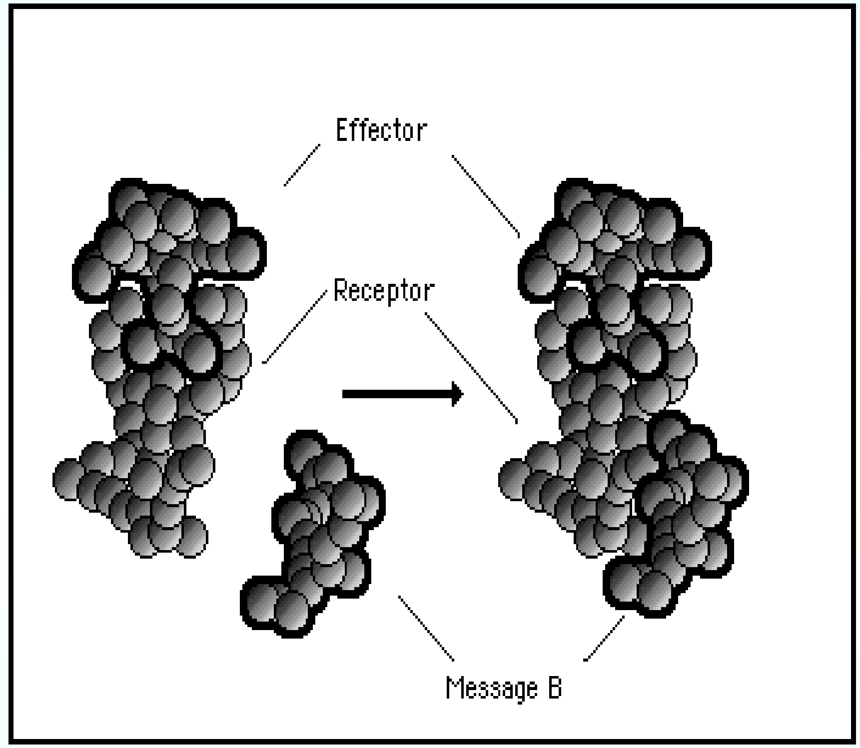
# Example Characteristics of SOAs

- Standard-based (XML related standards – strongly tagged)
- (Continued) Emphasis on data and information exchanges
- Emphasis on registries and mediation architectures
- Shifting from interfaces to contracts
- Technical and (…future) business negotiation
- Ontologies and other semantic aspects
- From "interface" to "interoperability architecture problem"
- New security challenges
- Allow (and require) new management solutions
- Mindset switch from "install and use" to "find and invoke"
- Very loose coupling

# Early SOA and Post SOA Models :>)




**Early SOA**
**First Diagram of the Internet**
**UCLA to CERN (ARPA)**

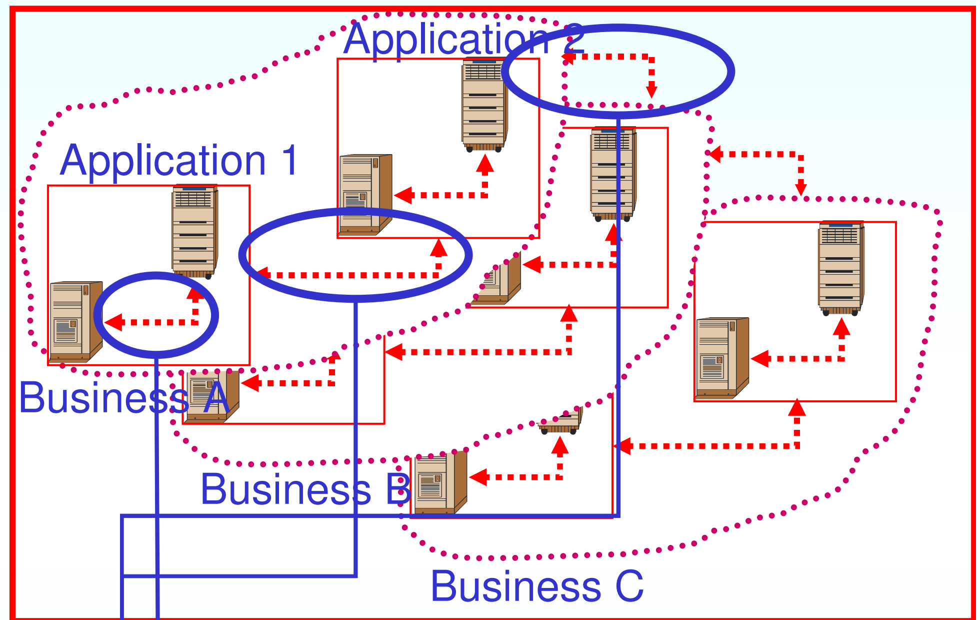


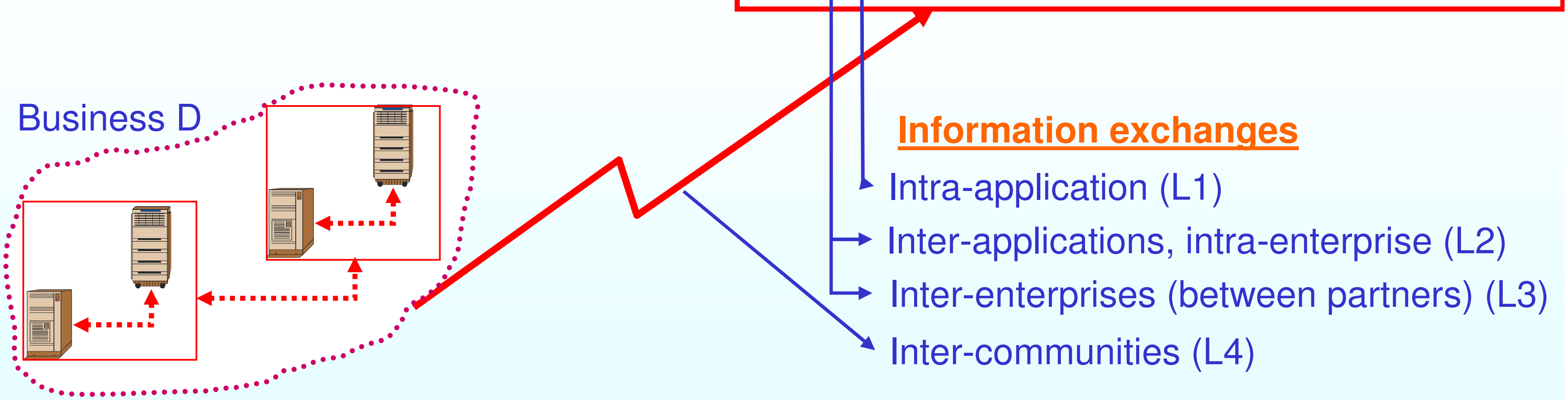

**Post SOA**
**Genetic Dispatching**
**w/Protein Messengers**

Source: http://www.stigmergicsystems.com/F

# COSM™ Levels of Information Exchange

Different levels of information exchanges typically require:
- different architectural approaches
- different technologies
- different standards



# Typical Coupling Characteristics

- **Intra-Application (L1)**: Tightly coupled applications
- **Inter-Applications, Intra-Enterprise (L2)**: Tightly coupled applications or loosely coupled applications (for example through messaging)
- **Inter-Enterprises, Intra-Community (L3)**: Loosely or Very Loosely coupled applications
  - Web services
- **Inter-Communities (Federated Enterprise Systems/Applications – L4)**: Very loosely coupled systems/applications
  - Web services

# Examples: L2-L3 Architectural Choices

## L2 : Inter-applications, intra-enterprise

- Quite coarse grained exchanges (can be less than document)
- Sub-transactional or transactional information exchanges

## L3 : Inter-enterprises

- Coarse-grained documents
- Transactional boundaries managed by individual enterprises
- Business transactions managed simply by compensation actions

HERZUM SOFTWARE

# Evolution: Computing Systems and Applications

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| | Decentralized Batch Systems | Centralized Batch Systems | Distributed Access Computing | End User Computing | Client/Server Computing | Web-Based Computing | Ubiquitous Computing |
| COMPUTER USAGE | Transistors | Mainframes | Minicomputers | PCs & LANS | Enterprise Networks | Internet & the Web | Wireless Internet, Web, Handheld & other Devices |
| | 1401 ON EVERY FLOOR | IBM 360 | DUMB TERMINALS | | L1, L2 | L3 | L4 |
| | TABULATING | SHARED RECORD KEEPING | ONLINE TRANSACTION PROCESSING & TIMESHARING | PERSONAL PRODUCTIVITY & GROUPWARE | ENTERPRISE RESOURCE PLANNING (ERP) | e-BUSINESS & COLLABO-RATIVE COMPUTING | WIRELESS COMPUTING<br>End user-driven e-business, Security & Privacy |
| | | Information Systems | | Personal & Workgroup Systems | Intra-Enterprise Systems | Inter-Enterprise Systems | Federations of Enterprises |
| | Pre-1965 | 1965-1975 | 1975-Present | 1982-Present | 1990- Present | 1996 Forward | 2001 Forward |

Adapted from: Peter Fingar

HERZUM SOFTWARE

# “Middleware Evolution”

**“Traditional” Integration**

Point to Point Integration → Bus-Based

**Web Services (Architecture & Technology Focus)**

Internal (L2) → External (L3 & L4)

**Web Services (Business focus)**

Process-Oriented (internal) → Process-Oriented (external)

# COSM™ Interoperability Reference Model

| |
|---|
| Development Life Cycle Aspects |
| Functional Reference Model |
| Semantics |
| Functional Interfaces |
| Structural Infrastructure |
| Technical Infrastructure |
| Technical Interfaces |

**(For more info, see free download at [www.herzumsoftware.com](www.herzumsoftware.com) or "Business Component Factory" book)**

HERZUM SOFTWARE

# Application-to-application within an Enterprise

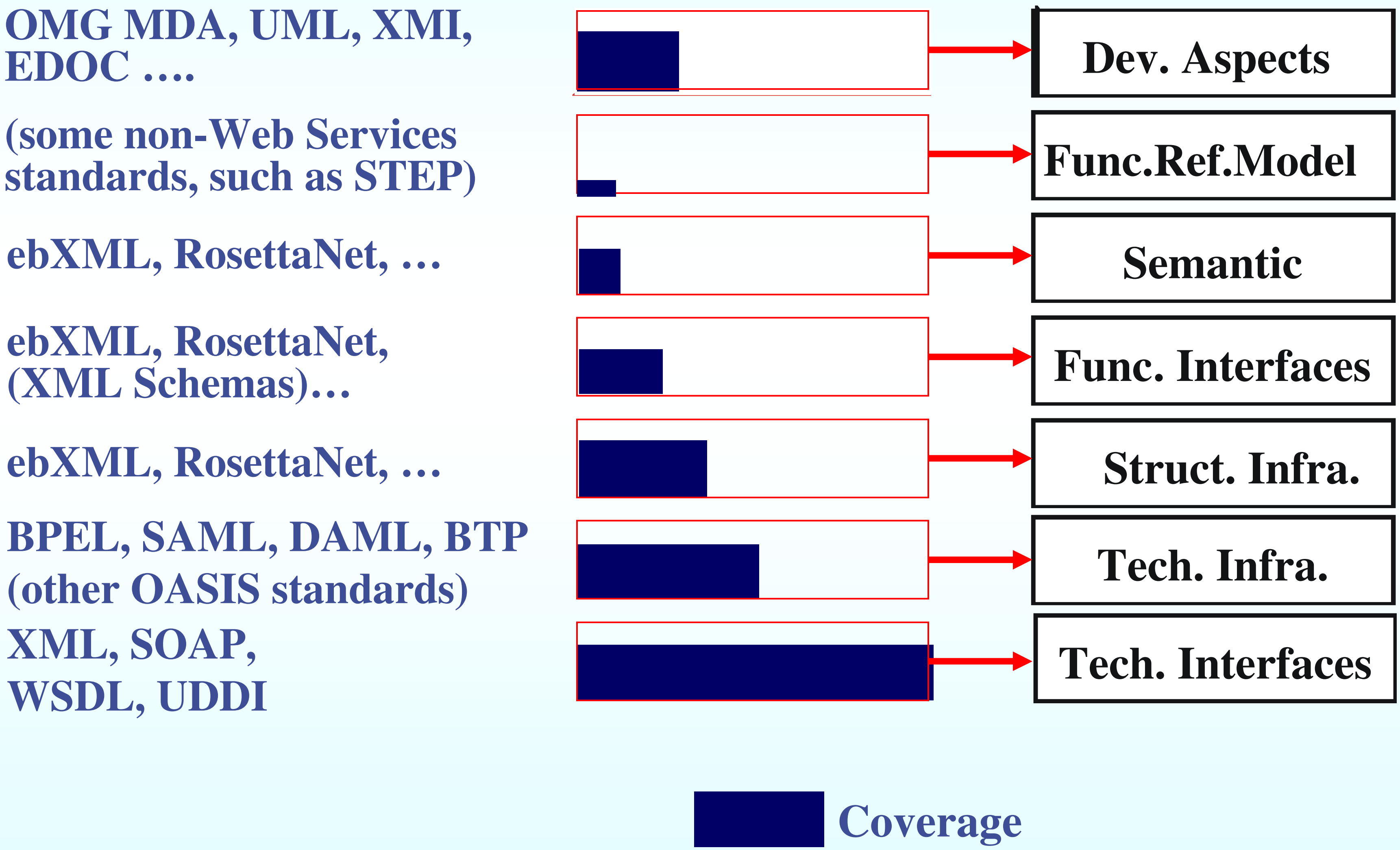
| | |
|---|---|
| Using same development environment | → Development Lifecycle Interfaces |
| Implicit | → Functional Reference Model |
| Agreed by phone | → Semantic |
| Defined by internal functional and component architects | → Functional Interfaces |
| Internally defined, tightly coupled architectural standards | → Structural Infrastructure |
| J2EE, .Net, Corba services and facilities | → Technical Infrastructure |
| RMI, Corba, .NET Remoting, (XML?) | → Technical Interfaces |

HERZUM SOFTWARE

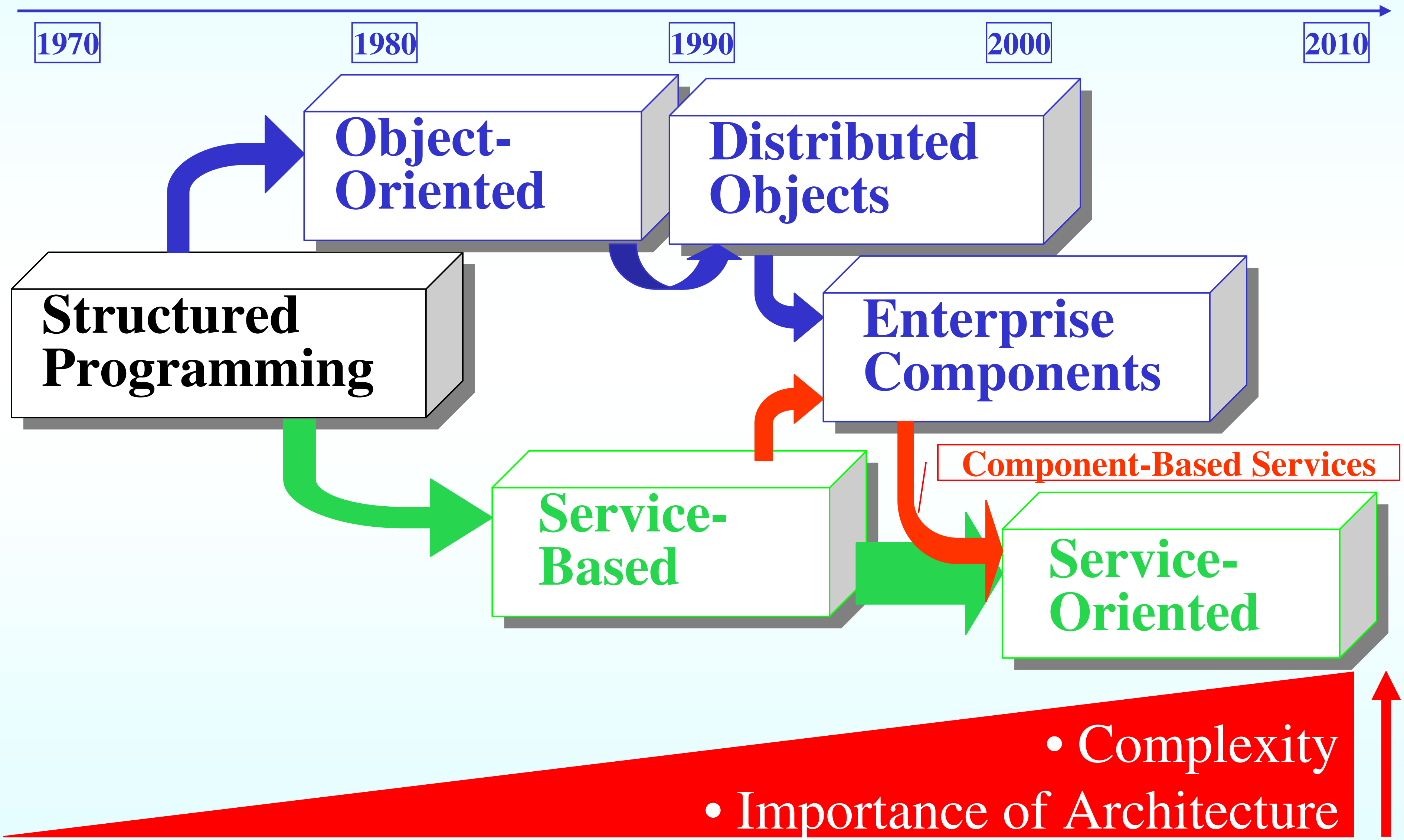
# Enterprise-to-Enterprise

| | |
|---|---|
| UML, MOF, XMI, … | Development Lifecycle Interfaces |
| Rarely exist, but should be standard | Functional Reference Model |
| Ontology | Semantic |
| Standard defined (e.g. webservices) | Functional Interfaces |
| Standard defined, loosely coupled architectural standards | Structural Infrastructure |
| ebXML | Technical Infrastructure |
| XML, HTTP, SOAP | Technical Interfaces |

# Gaps in Web Services Standards Coverage



# Architectural Styles



HERZUM SOFTWARE

# Brief History of Time: Components

1970 | 1980 | 1990 | 2000

**Structured Programming**

Cobol, Pascal, Ada,
One Technology,
Structured Methodologies

**Object-Oriented**

C++, Eiffel, One Technology, OO Methodologies, Very Fine Granularity, Tight Coupling

**Distributed Objects**

CORBA 1, Java, Only Tech. Interop., UML, Fine Granularity, Tight Coupling. Low complexity (no real transactions, security)

**Enterprise Components**

J2EE, CORBA 3, .Net, Coarse Granularity Possible, XML, Loose Coupling, MDA

HERZUM SOFTWARE

# Brief History of Time: Web Services

1970 | 1980 | 1990 | 2000 | 2010

**Structured Programming**

Cobol, Pascal, Ada,
One Technology,
Structured Methodologies

**Service-Based**

TPM (Tuxedo, ACMS, CICS, …), Messaging, n technologies, System Granularity, Loose Coupling

Internet, EAI, eBusiness, …

**Service-Oriented**

Web Services, Mediation Architectures, BTP, System Granularity, **Very** Loose Coupling

# Components v/s Services

| | Components | Services |
|---|---|---|
| Architectural Perspective | The internal asset of a system (not necessarily shown outside, externals can be Web services or not) | What is seen externally to a system (internals can be components or not) |
| Deployment Model | Software is physically deployed. “Install and use” | Software “exists” somewhere. “Connect and use” |
| Levels of Information Exchange | Mostly within enterprise | Mostly across enterprises |
| Coupling | Fairly loose. Based on internal standards | Very loose. Based on industry standards |
| Communication | Enterprise-based Protocols (like RMI or .NET remoting) | Internet-based Protocols (like XML over SOAP) |

# Service-Based Architectural Style

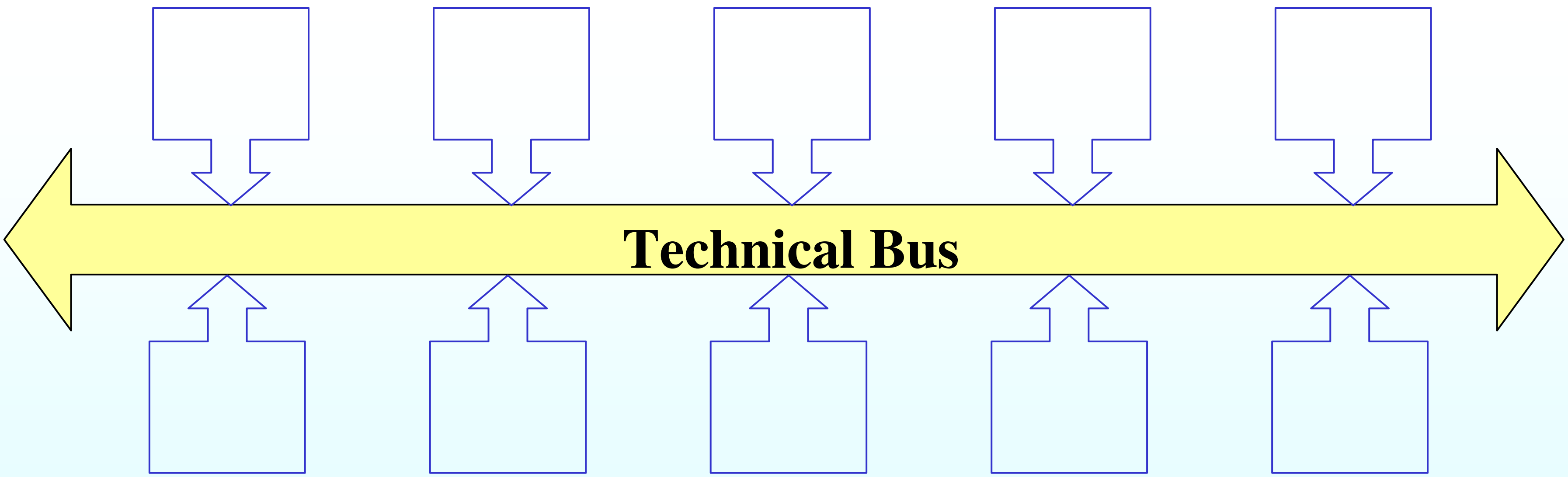
**Main Characteristics**

- Focus on providing distributed interfaces to existing systems, and connecting them to each other
  - Often through bus-based or hub-and-spoke technology infrastructure
  - Often through simple (technical) mapping to a given distributed technology
- Granularity of "service implementation" is often large monolithic systems
- Individual services are usually transactional (TPM meaning)
- Typically no attempt at providing "autonomous" implementations of services, or reducing dependencies, or associating islands of data
- No layering

# Nature of federation in service-based

**Each <u>system</u>**

- **As black-box (run-time, development-time)**
- **Heterogeneous technologies. Each system manages it own transactions**



# Service Based Platform
# The Component Execution Environment

- A factory setup element providing an architectural framework and supporting tools
- An architectural framework is a technology independent platform model defined to
  - Provide a unique Service Oriented and Component Based Architectural Style across the system
  - Address the critical concerns for constructing enterprise class systems
  - Integrate best of breed ingredient technologies to meet the changing market needs
- The supporting tools enable functional development teams to
  - jump start component development
  - focus on domain capabilities rather than infrastructure development
  - maintain alignment with an overall architectural blueprint

# Platform & Functional Web Services

- **Orchestrated Web Services**
  - **Providing actual "business-level" (complex) Web Services: getAllCustomerAccounts, reserveTravel**
- **Element Web Services**
  - **getCustomerAddress, getCustomerAccount, …**
- **Utility Web Services**
  - **Authorization & authentication, unit of measure conversion, number generation (inside an enterprise or a system)**
- **Web Services Technical Infrastructure High level**
  - **Distributed Registry Service, Security Infrastructure, Ontology Services, Access Rights Manager, Service Modeler, Business Process & Workflow Modeler, QoS …("Infrastructure Web Services")**
- **Web Services Technical Infrastructure Low level**
  - **Simple Lookup & Register**
  - **Web Services-Specific Network layer**

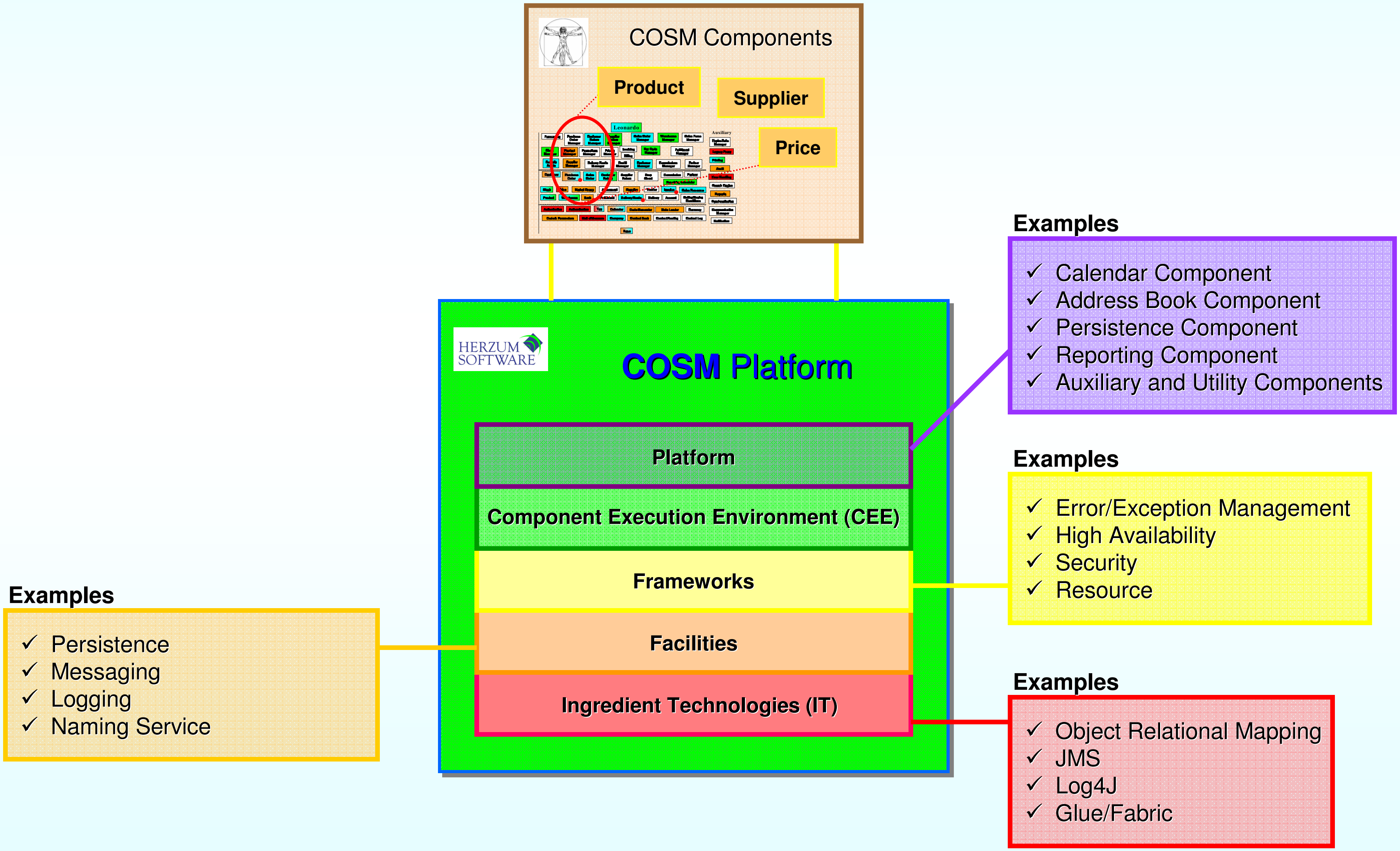
| |
|---|
| Orchestrated Web Services |
| Element Web Services |
| Utility Webservices |
| Tech. Infra. High Level |
| Tech. Infra. Low Level |

# COSM™ Platform Elements

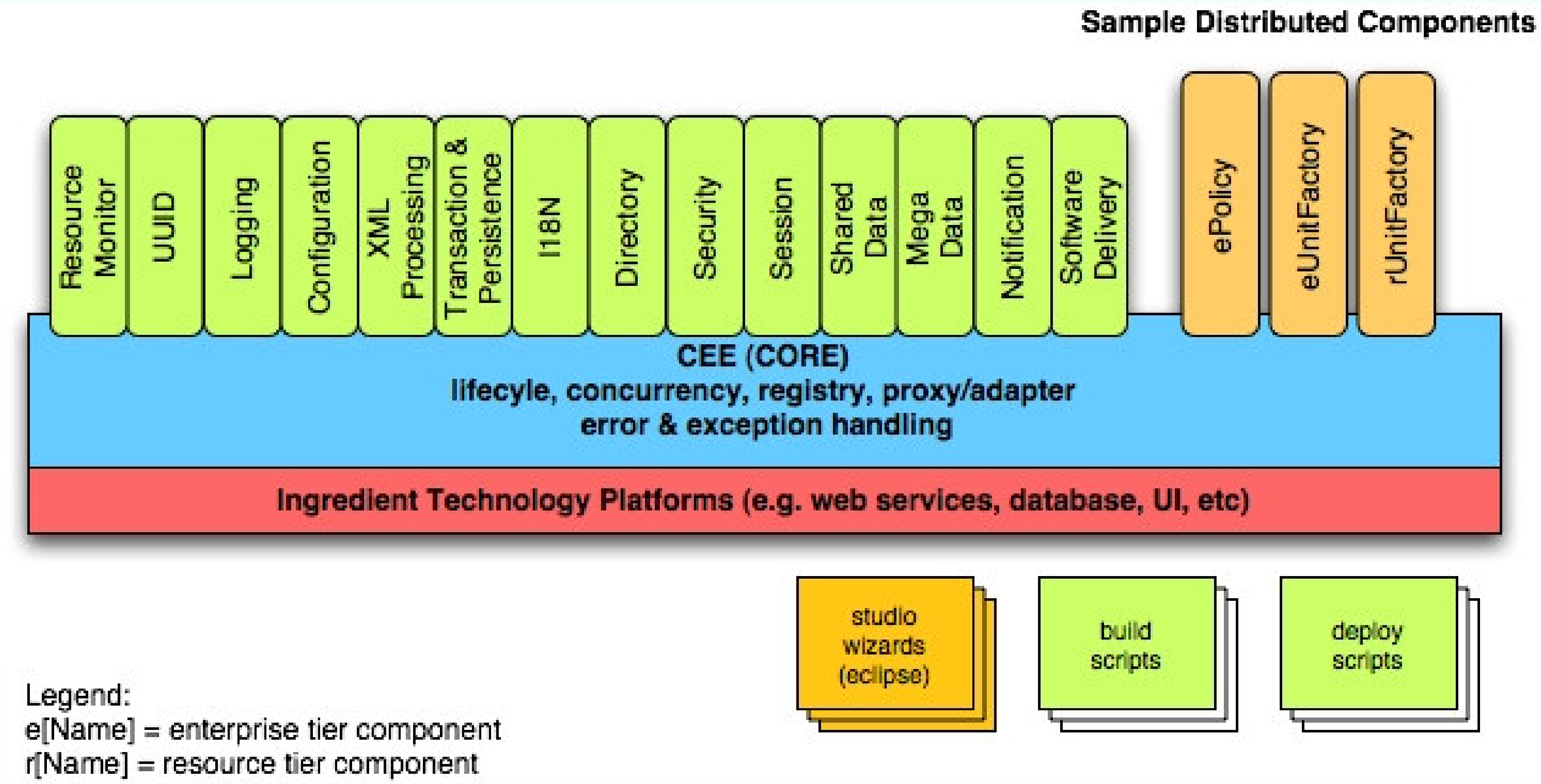
**The platform is composed of:**

- **CEE Kernel. Responsible for:**
  - **Component Lifecycle Management**
  - **Service Discovery**
  - **Load Balancing**
  - **Fault Tolerance**
  - **Providing POJO development environment**
  - **Security**
  - **Component and CEE administration**
- **CEE Facilities. Responsible for:**
  - **Exposing simple, technology agnostic interfaces abstracting the underlying technologies**
  - **Removing the need for the functional developers to understand technology specific nuances**
- **Utility and Auxiliary Components. Responsible for:**
  - **Common base functionality required by enterprise class applications. Examples:**
    - ***Printing***
    - ***Reporting***
    - ***Calendaring***
    - ***Address Book***
    - ***Security***

# Inside the COSM™ Platform



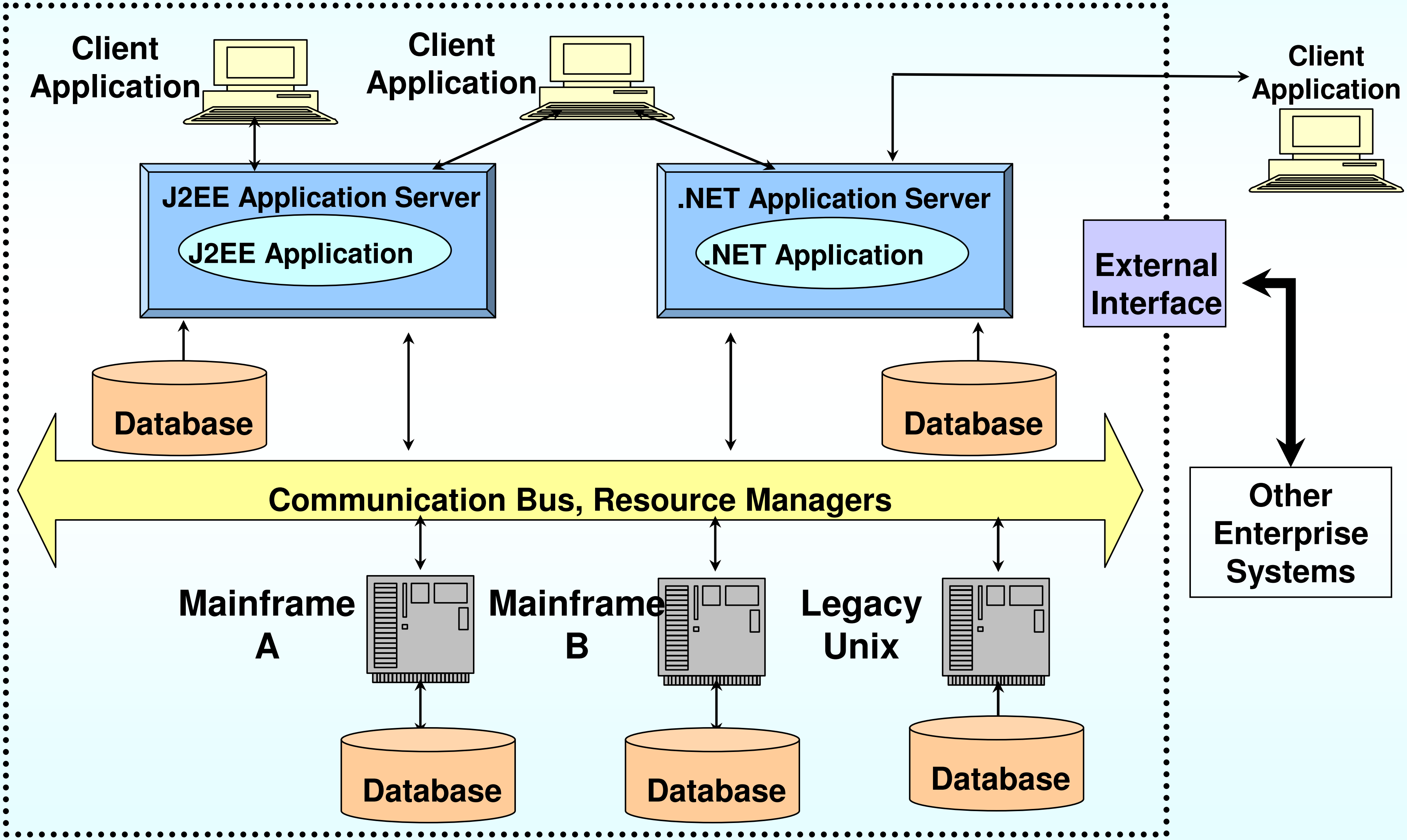
# The COSM™ Platform: CEE



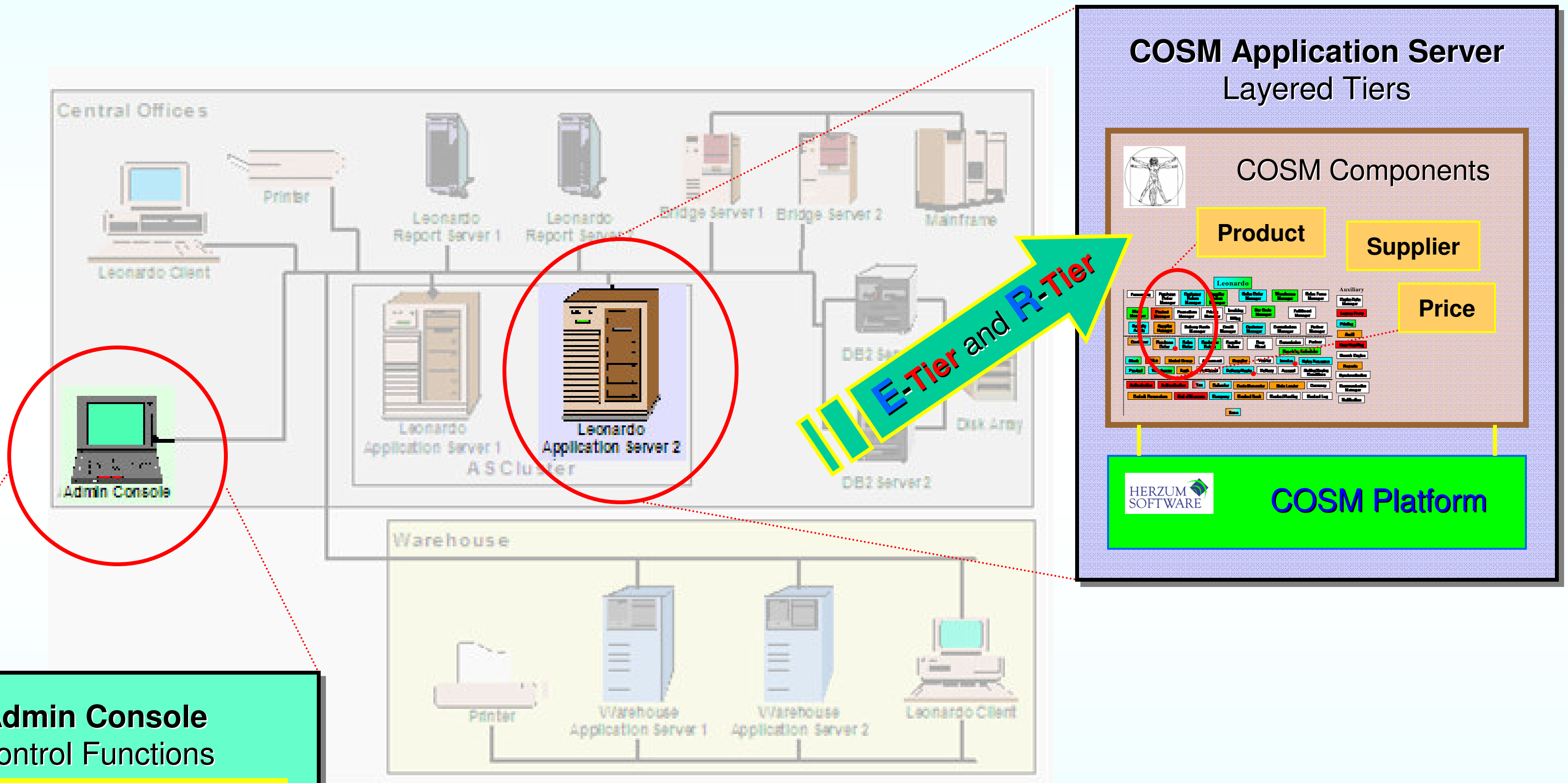
# Sample COSM CEE™ Features

| Feature | Feature Description |
|---|---|
| **Logging** | Enables distributed collection of log messages generated by the CEE, the platform components, the applications |
| **Directory** | Services and Components are registered and deregistered through the directory – supports autodiscovery and service metadata |
| **Mega Data** | Enable the reliable and progressive delivery of large amounts of data |
| **Shared Data** | Acts as a P2P shared memory space for the distributed CEE and supports replication of state for pervasive services like session management |
| **Configuration Management** | Allows for all components of the CEE to access system and component level centrally managed configuration data |
| **Smart Proxy / Adapter** | The smart proxy framework works behind the scenes, providing enhancements to each CEE service like auditing, tracing, etc. |
| **Async Invocation** | Provides the option of invoking service calls asynchronously or fire-and-forget |
| **Load Balancing** | Monitors instances of the CEE for critical resource utilization. When querying for services, the CEE instance with the most available resources is used. |
| **Failover** | The failover mechanism allows for session-level and invocation-level errors to be trapped and handled |
| **UUID** | Returns a globally unique key (URIs) used to identify object instances within a running cluster of CEE(s) |
| **XML Processing** | Handles intelligent Java to XML marshalling and un-marshalling |
| **Studio** | A plug-in to the Eclipse IDE to allow for component specification and the automation of some CEE development tasks (i.e., asynchronous interface generation.) |
| **Session Management** | Support the creation, distribution, and access to session-level data (e.g. user profiles) |
| **Security** | Provides for the centralized definition and distributed enforcement of a system defined security policies |
| **Transaction** | Support implicit and explicit transaction management |
| **Persistence** | Provides means for the generation and runtime management of object-relational mappings |
| **Internationalization** | Enables the central specification of I18N resource bundles and distributed runtime access on the basis of deployment configuration |

# Example: “Ingredient Technologies”



# COSM™ Deployment Architecture




**Admin Console**
Control Functions

- ☑ Deploy New Components
- ☑ Start/Stop Components
- ☑ View Running Components
- ☑ View Log Files

# COSM™ Platform
# Integration of Open Source Products

- ***Technology perspective:*** **The COSM™ platform is pre-selecting and integrating the ingredient technologies from the fragmented Open Source product community and providing an application development factory and large-scale runtime environment using a mature component based development approach.**

***Open Source***

- ***Liferay (Portal)***
- ***Lenya (Content Management)***
- ***CAS – (Security)***
- ***Joram (JMS - async messaging)***
- ***Log4j (logging)***
- ***Quartz (scheduling)***
- ***Jython (scripting)***
- ***Hibernate (ORM)***
- ***Struts (UI)***
- ***JBoss (EJB Server)***
- ***Ant + Cruise Control (build env)***
- ***CVS (change management)***
- ***Atlassian Jira (issue management)***
- ***Sleepycat: Efficient caching***
- ***Drools (Rule Engine )***
- ***BSF (Bean Script Framework)***
- ***JUnit (Unit testing framework)***
- ***Others***

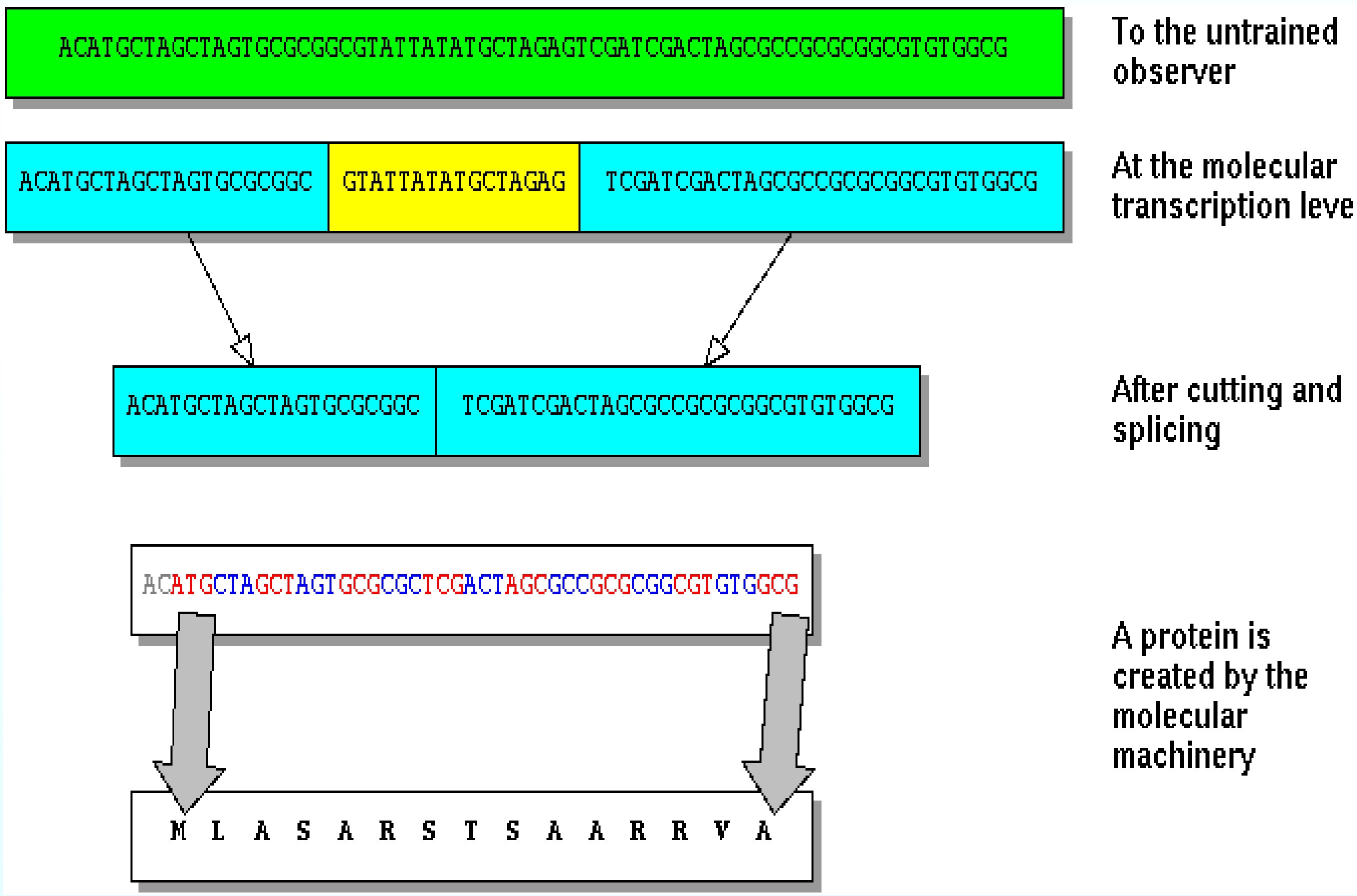
# CEE Addresses the Application XFRs Contract

- CEE provides the technical infrastructure required to meet the COSM extra functional requirements for
  - Scalability
  - Fault Tolerance
  - Performance
- CEE accomplishes these goals via an integrated strategy that blends
  - Federated Clustering
  - Fine Grained Resource Load Monitoring
  - Adaptive (Bind + Invoke) Load Balancing
  - Service Level Heart Beat Monitoring
  - Autonomous Provider Selection
  - SLA based Service Failover
  - State Replication

# Looking Forward: Gene Sequenced Metadata – Our Future Semantic Stores?

## What is a protein-encoding sequence?

A portion of the DNA provides a "coded message" that describes what pieces ought to be used for making a protein. After the raw message is sent, it is decoded by the protein-construction machinery. The decoding is done using the Genetic Code.



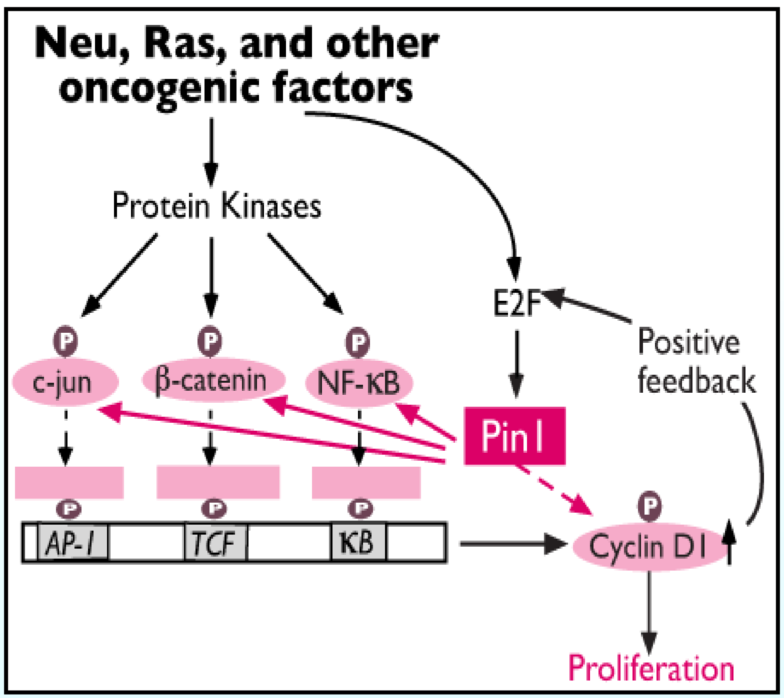

Source: http://ccgb.umn.edu/~crow/docs/gf-principles3/protein-synth.html

## *Looking Forward - Molecular Middleware*
## *Our Future IRM Protocols?*

**"Key Relay Protein Shapes Cancer Message**

***Refolding of Messenger Molecules Needed for Breast Tumor Development***

A tiny enzyme able to latch onto proteins and change their shape plays a critical and surprising role in promoting some of the most belligerent forms of breast cancer…Among the fastest growing are those that produce an overabundance(1) of the receptor Her2/Neu. Once activated, Her2/Neu ***sends a signal, via a series of molecular messengers, to an agent lying in wait inside the nucleus that tells the cell to enter the cell cycle***. Though researchers have identified some of the middlemen, it is not clear how the message is delivered to the cell cycle gatekeeper, cyclin D1" Source: http://focus.hms.harvard.edu/2004/July16_2004/oncology.html (1) Molecular spam?




- **COSM™'s Interoperability Reference Model**
- **Development Life Cycle Aspects**
- **Functional Reference Model**
- **Semantics**
- **Functional Interfaces**
- **Structural Infrastructure**
- **Technical Infrastructure**
- **Technical Interfaces**

# Conclusions

- Architectural Reference Models provide key abstractions for positioning middleware solutions
- Information Exchange Models provide context for understanding tradeoffs in levels of coupling and binding models
- The Interoperability Reference Model describes the key architectural layers involved in integrating a service based middleware solution
- The evolution of Architectural Styles into a service based approach has the benefits of a federated architectural model, but with corresponding architectural complexity
- Commercial component and service based platforms (e.g. COSM™ CEE) are emerging mature for certain technology aspects that provide important platform services over various infrastructures
- Reference models provides maps to guide us in understanding future middleware and SOA approaches

# Questions?

**Herzum Software**
**175 N. Franklin St., Suite 301**
**Chicago, IL, 60606  USA**

**"*Business Component Factory*", Herzum & Simms, John Wiley & Sons, 1999**

**information@herzumsoftware.com**

**wnadal@herzumsoftware.com**

**www.herzumsoftware.com**